

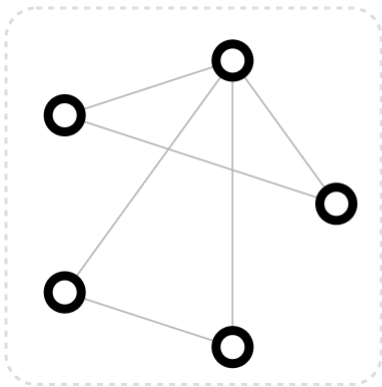
Graph Neural Networks

Giuseppe Serra

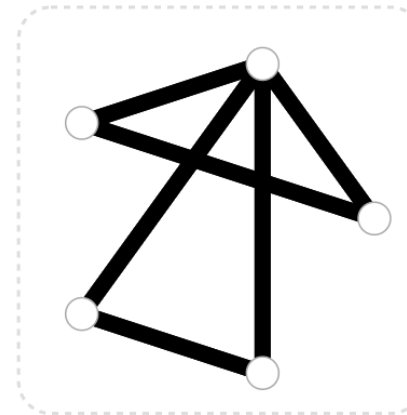
University of Udine

Graph Schema

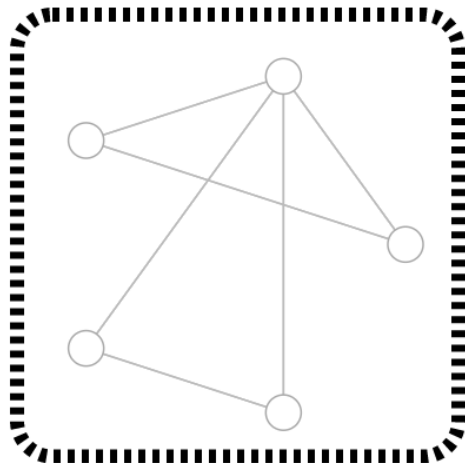
A graph represents the relations (*edges*) between a collection of entities (*nodes*).



- V** Vertex (or node) attributes
e.g., node identity, number of neighbors
- E** Edge (or link) attributes and directions
e.g., edge identity, edge weight
- U** Global (or master node) attributes
e.g., number of nodes, longest path



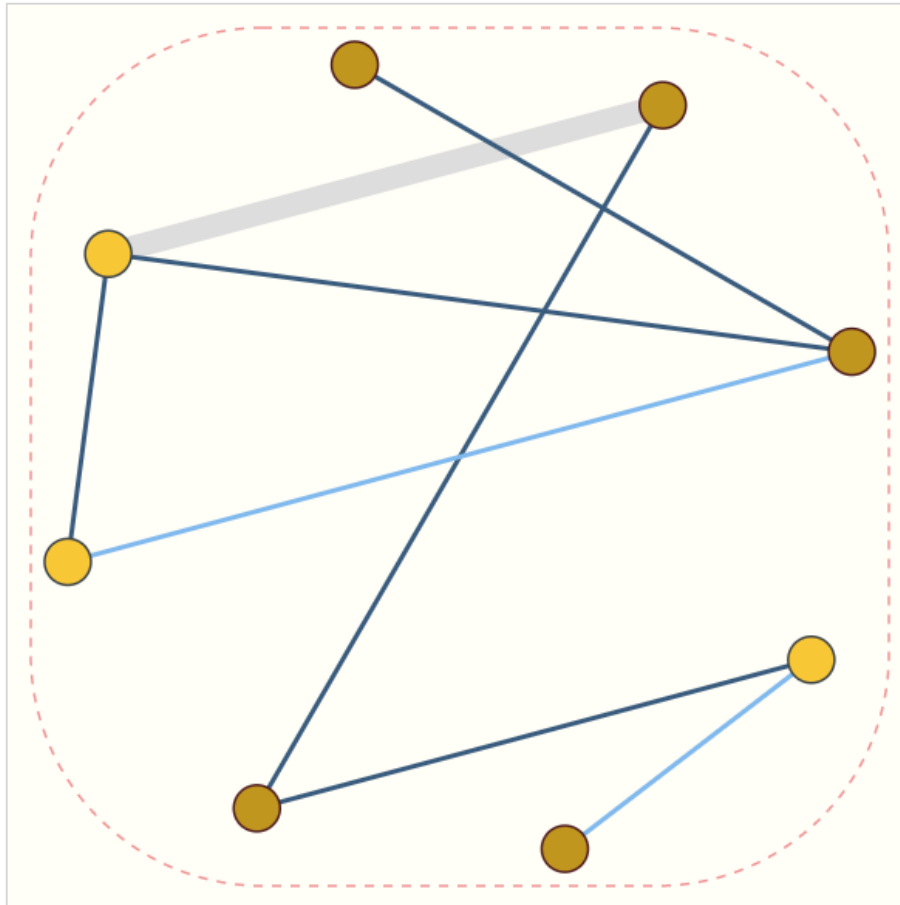
- V** Vertex (or node) attributes
e.g., node identity, number of neighbors
- E** Edge (or link) attributes and directions
e.g., edge identity, edge weight
- U** Global (or master node) attributes
e.g., number of nodes, longest path



- V** Vertex (or node) attributes
e.g., node identity, number of neighbors
- E** Edge (or link) attributes and directions
e.g., edge identity, edge weight
- U** Global (or master node) attributes
e.g., number of nodes, longest path

Graph Schema

A Graph representation



Nodes

[0, 1, 1, 0, 0, 1, 1, 1]

Edges

[2, 1, 1, 1, 2, 1, 1]

Adjacency List

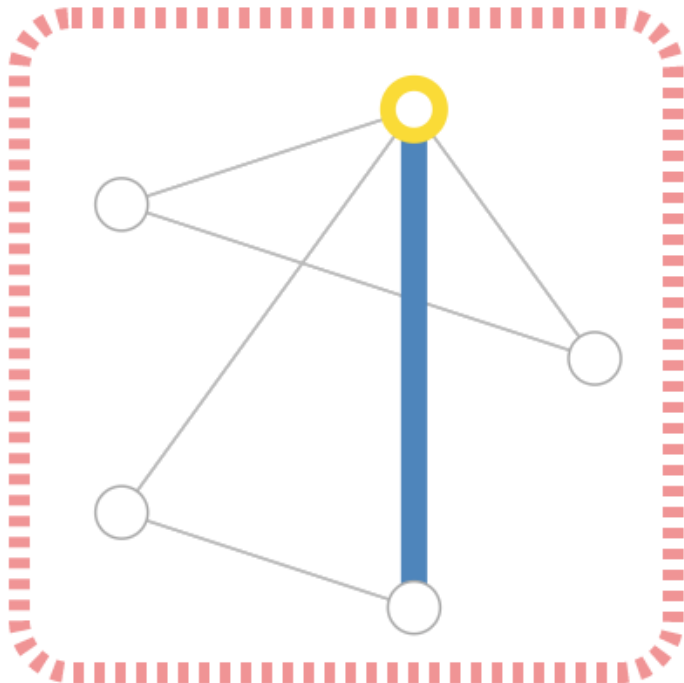
[[1, 0], [2, 0], [4, 3], [6, 2],
[7, 3], [7, 4], [7, 5]]

Global

0

Graph Schema

To further describe each node, edge or the entire graph, we can store information in each of these pieces of the graph.



Vertex (or node) embedding



Edge (or link) attributes and embedding



Global (or master node) embedding

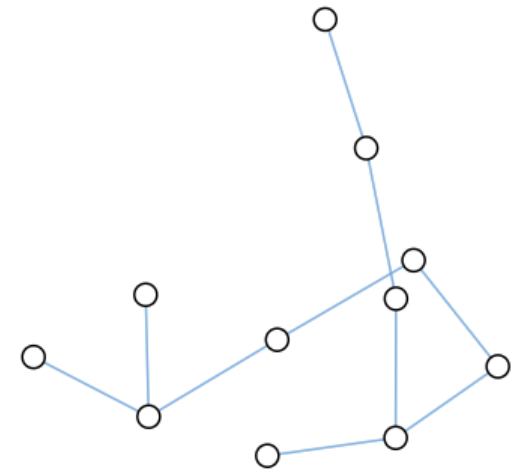
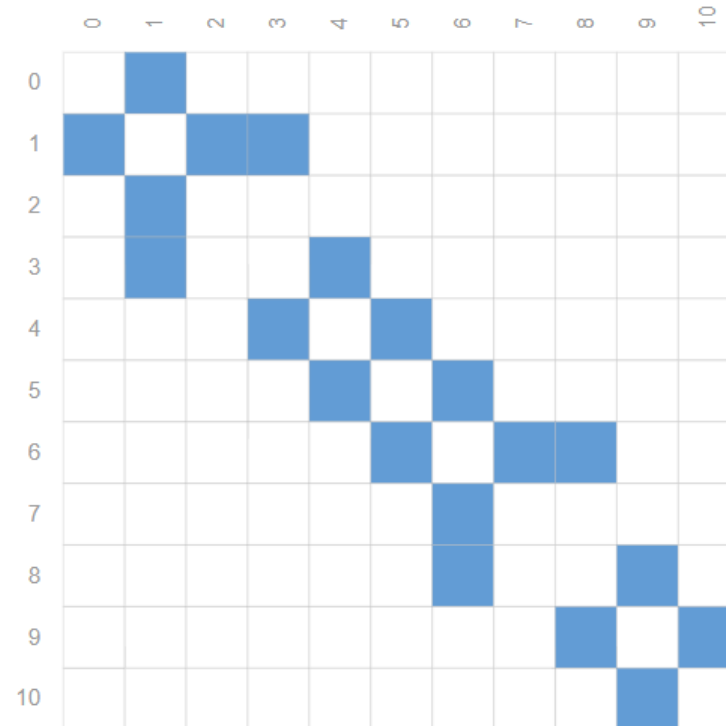
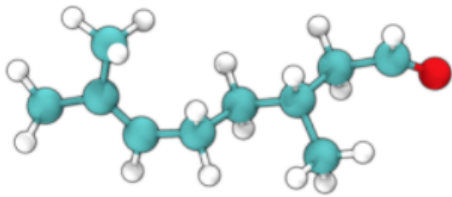


Data and Graph representation

Molecules as Graphs

Molecules are the building blocks of matter and are built of atoms and electrons in 3D space.

Here a **citronellal** molecule



Images as Graphs

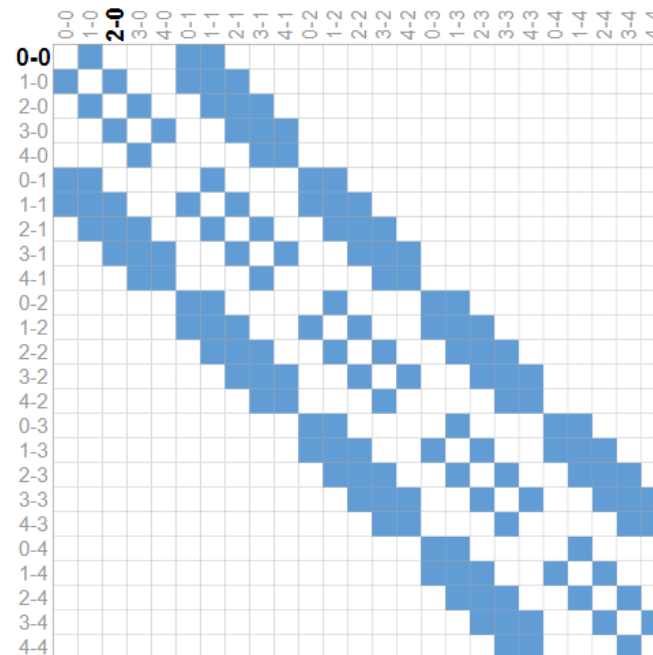
We typically think of **images as rectangular grids with image channels**, representing them as arrays (e.g., 244x244x3 floats).

Another way to think of images is as graphs with regular structure, where each **pixel represents a node** and is **connected via an edge to adjacent pixels**.

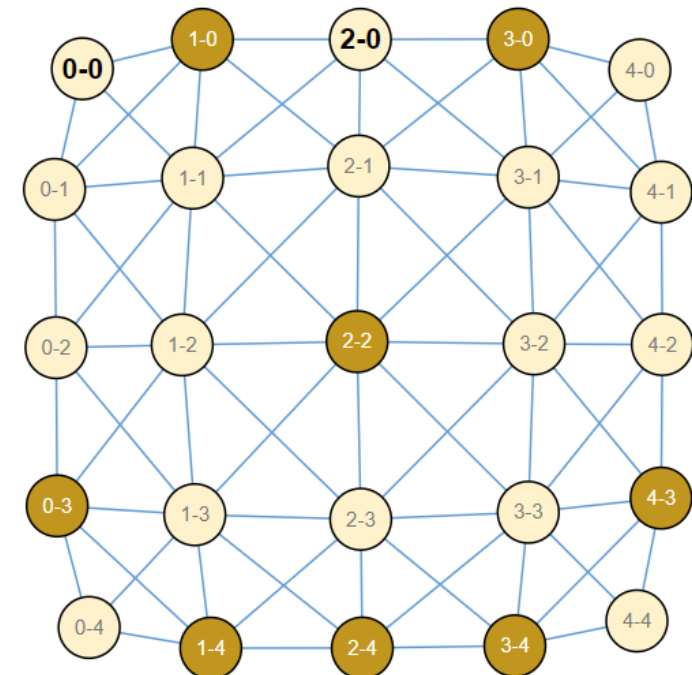
Each **non-border pixel has exactly 8 neighbours**, and the **information stored at each node is a 3-dimensional vector representing the RGB value of the pixel**.

0-0	1-0	2-0	3-0	4-0
0-1	1-1	2-1	3-1	4-1
0-2	1-2	2-2	3-2	4-2
0-3	1-3	2-3	3-3	4-3
0-4	1-4	2-4	3-4	4-4

Image Pixels



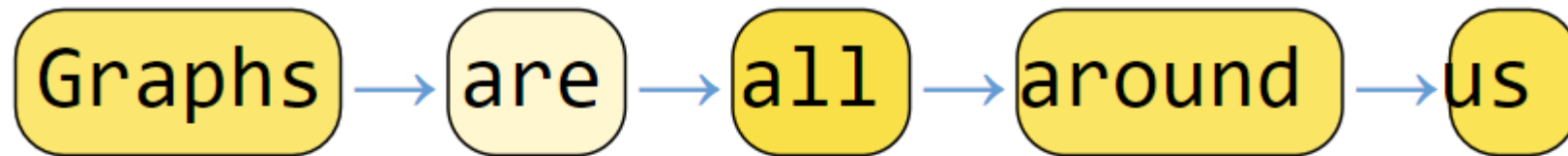
Adjacency Matrix



Graph

Text as Graphs

We can digitize text by associating indices to each character, word, or token, and representing text as a sequence of these indices. This creates a simple directed graph, where each character or index is a node and is connected via an edge to the node that follows it.



	Graphs	are	all	around	us
Graphs		■			
are			■		
all				■	
around					■
us					

Social Networks as Graphs

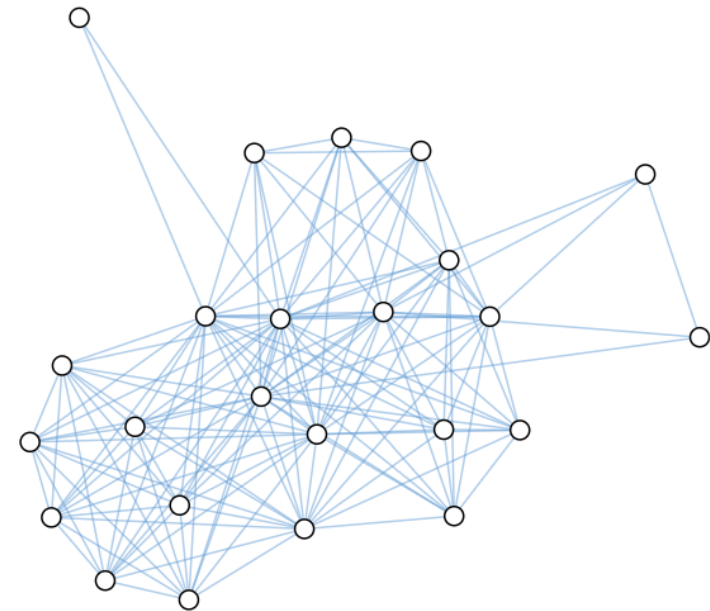
Social networks are tools to **study** patterns in collective **behaviour of people, institutions and organizations**.

We can build a **graph representing groups of people by modelling individuals as nodes, and their relationships as edges**.

Here an example of the play “Othello”



	Bianca	Brabantio	Cassio	Clown	Desdemona	Duke	Emilia	Gentleman	Gentleman.1	Gentleman.2	Gentleman.3	Gratiano	Iago	Lodovico	Messenger	Montano	Musician.1	Officer	Othello	Roderigo	Sailor	Senator	Senator.1	Senator.2
Bianca																								
Brabantio																								
Cassio																								
Clown																								
Desdemona																								
Duke																								
Emilia																								
Gentleman																								
Gentleman.1																								
Gentleman.2																								
Gentleman.3																								
Gratiano																								
Iago																								
Lodovico																								
Messenger																								
Montano																								
Musician.1																								
Officer																								
Othello																								
Roderigo																								
Sailor																								
Senator																								
Senator.1																								
Senator.2																								



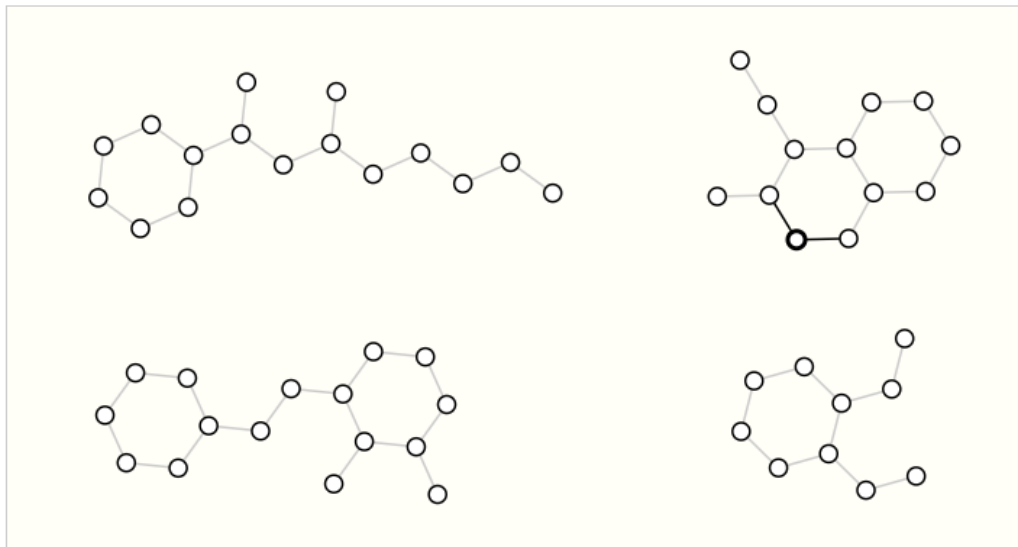
Prediction tasks on Graphs

Graph-level Task

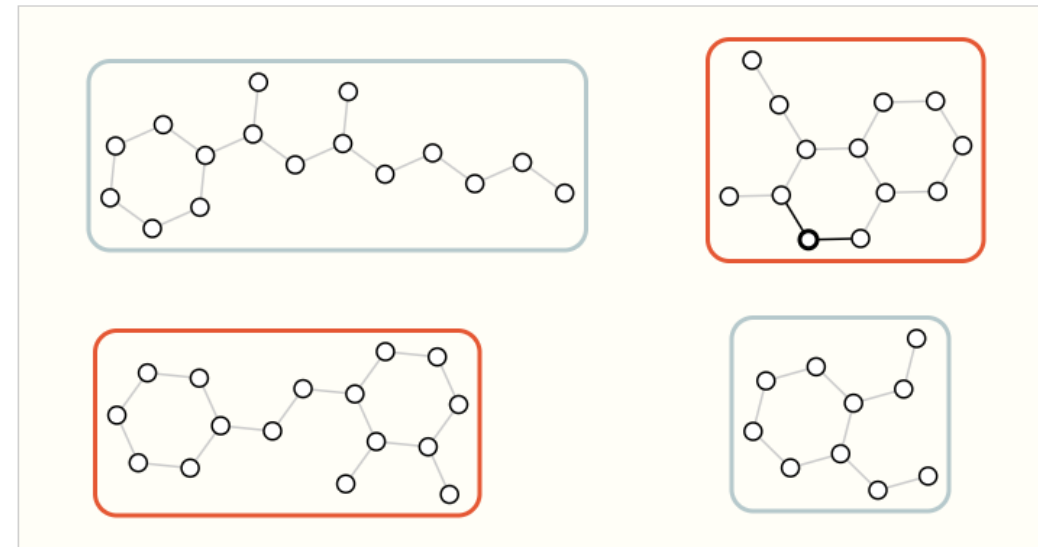
The goal is to predict the property of an entire graph.

For example, for a molecule represented as a graph, we **might want to predict what the molecule smells “pungent” or not.**

Or if a graph contains two rings:



Input: graphs



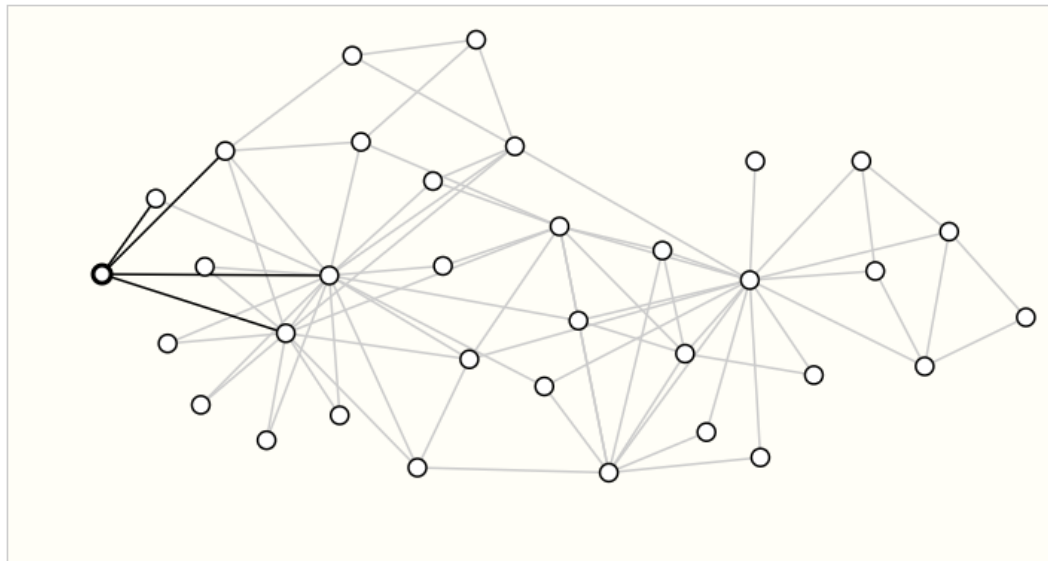
Output: labels for each graph, (e.g., "does the graph contain two rings?")

Node-level Task

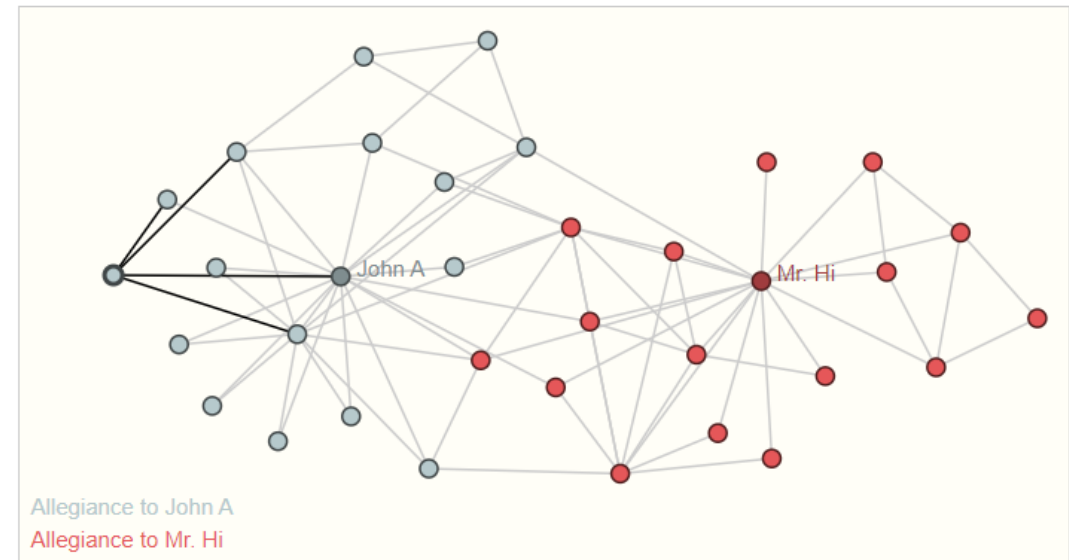
Node-level tasks are concerned with predicting the identity or role of each node within a graph.

A classic example of a node-level prediction problem is **Zach's karate club**.

As the story goes, a feud between **Mr. Hi (Instructor)** and **John H (Administrator)** creates a schism in the karate club. The **nodes** represent individual **karate practitioners**, and the **edges** represent **interactions between these members outside of karate**. The **prediction problem** is to classify whether a **given member becomes loyal to either Mr. Hi or John H**, after the feud. In this case, distance between a node to either the Instructor or Administrator is highly correlated to this label.



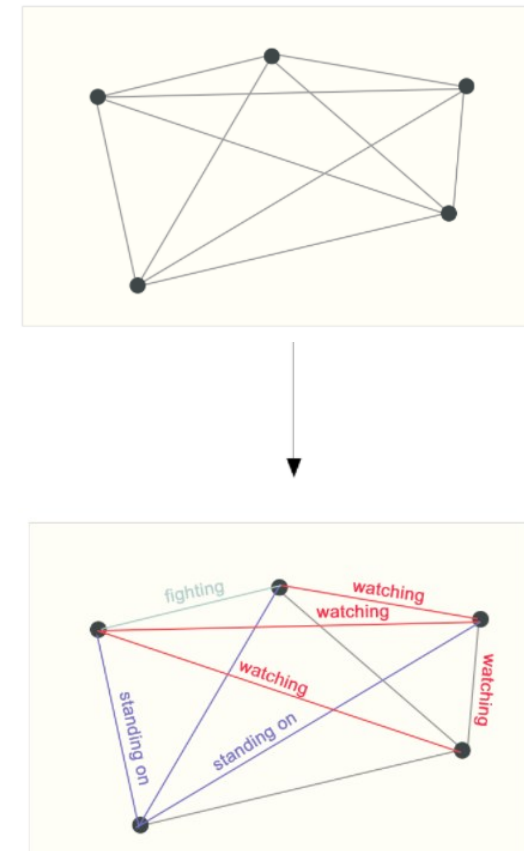
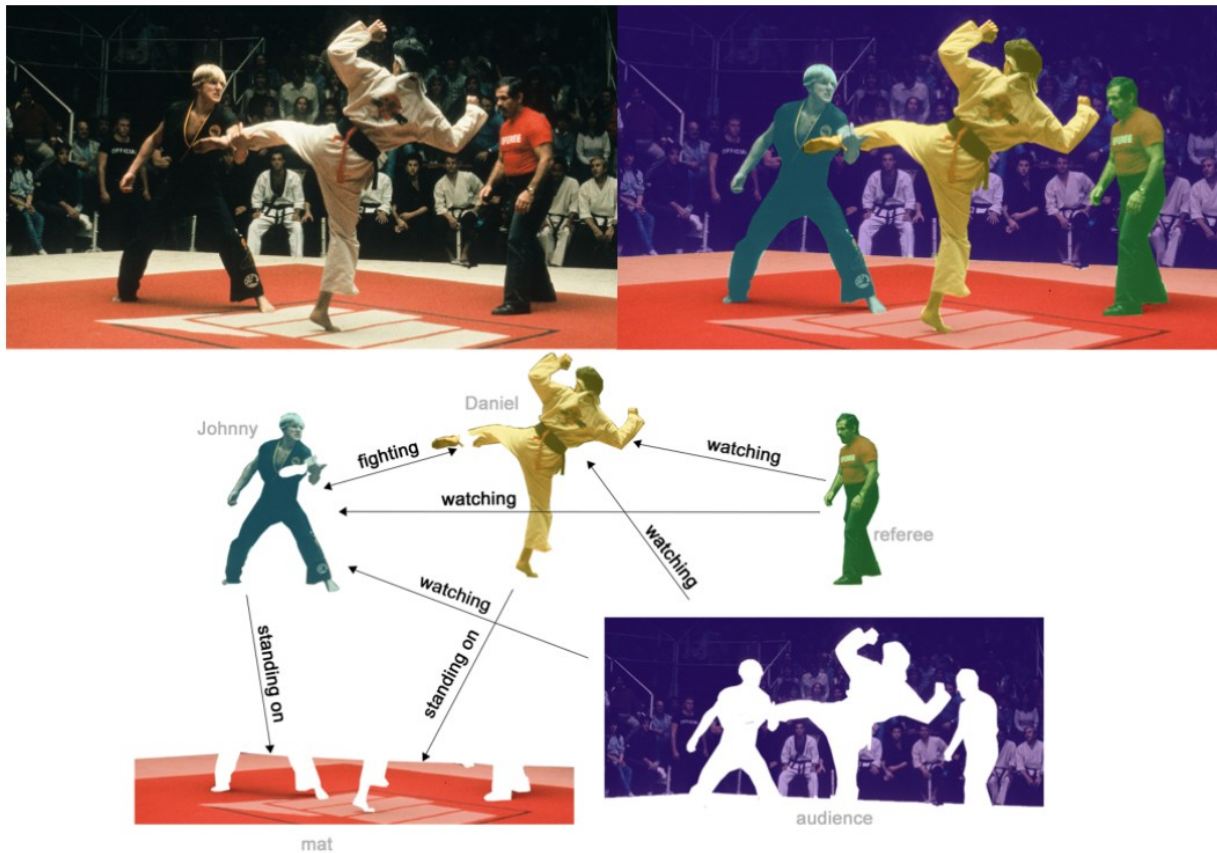
Input: graph with unlabeled nodes



Output: graph node labels

Edge-level Task

One example of **edge-level inference** is in **image scene understanding**. Beyond identifying objects in an image, deep learning models can be used to predict the relationship between them.



Graph Neural Networks

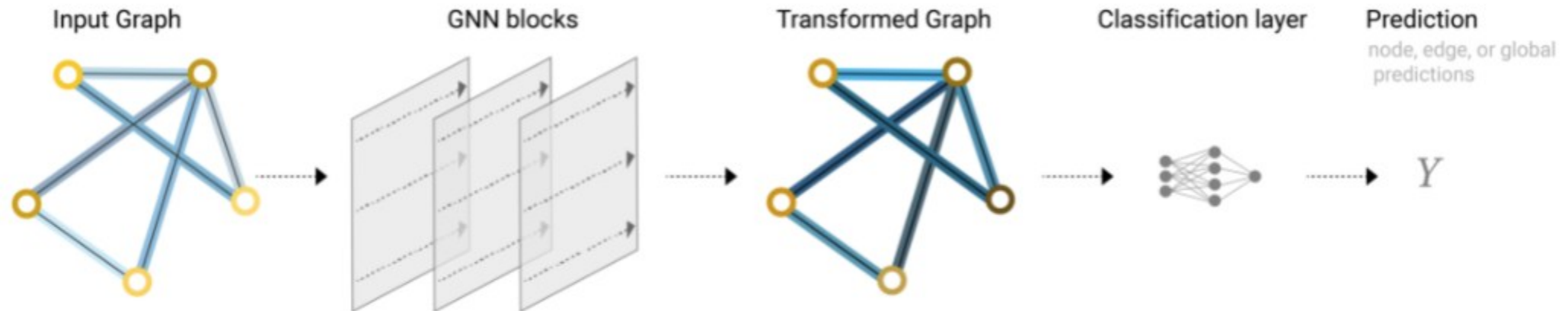
Graph Neural Networks

We're going to build GNNs using the “message passing neural network” framework.

GNNs adopt a “graph-in, graph-out” architecture meaning that these model types accept a graph as input, with information loaded into its nodes, edges and global-context, and progressively transform these embeddings, without changing the connectivity of the input graph.

As is common with neural networks modules or layers, we can stack these GNN layers together.

Here an example of an end-to-end prediction task with a GNN Model



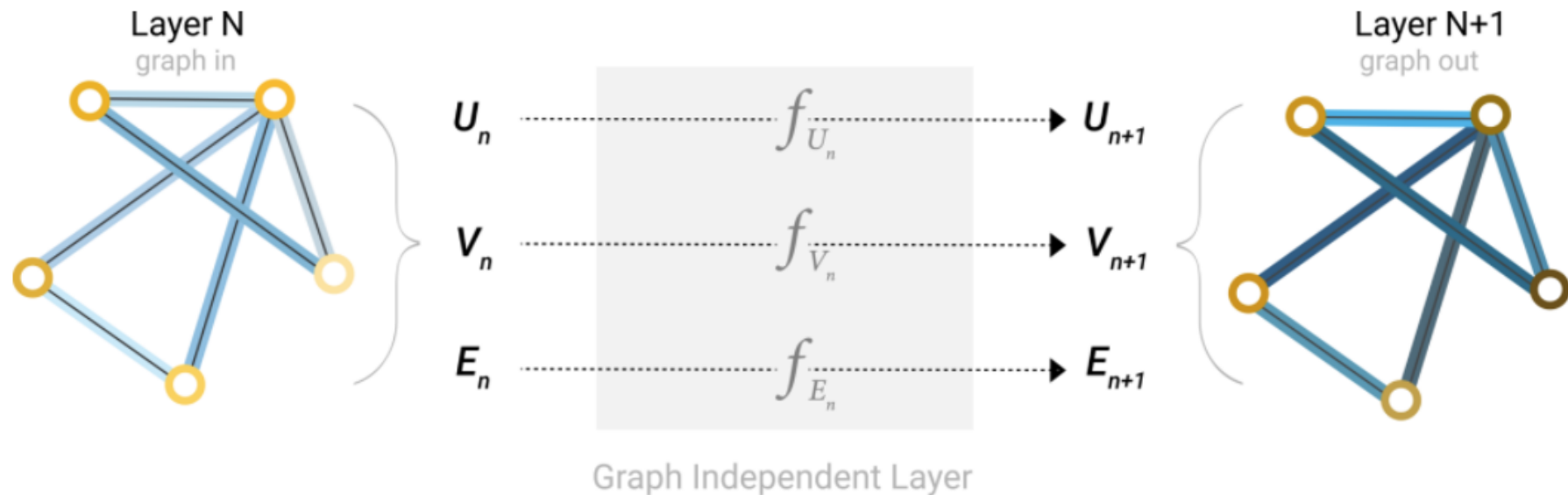
An end-to-end prediction task with a GNN model.


The simplest GNN

We will start with the simplest GNN architecture, one where we learn new embeddings for all graph attributes (nodes, edges, global), but where we do not yet use the connectivity of the graph.

This GNN uses a separate multilayer perceptron (MLP) (or your favourite differentiable model) on each component of a graph; we call this a GNN layer.

For **each node vector**, we **apply the MLP and get back a learned node-vector**. We do the same for each edge, learning a per-edge embedding, and also for the global-context vector, learning a single embedding for the entire graph.

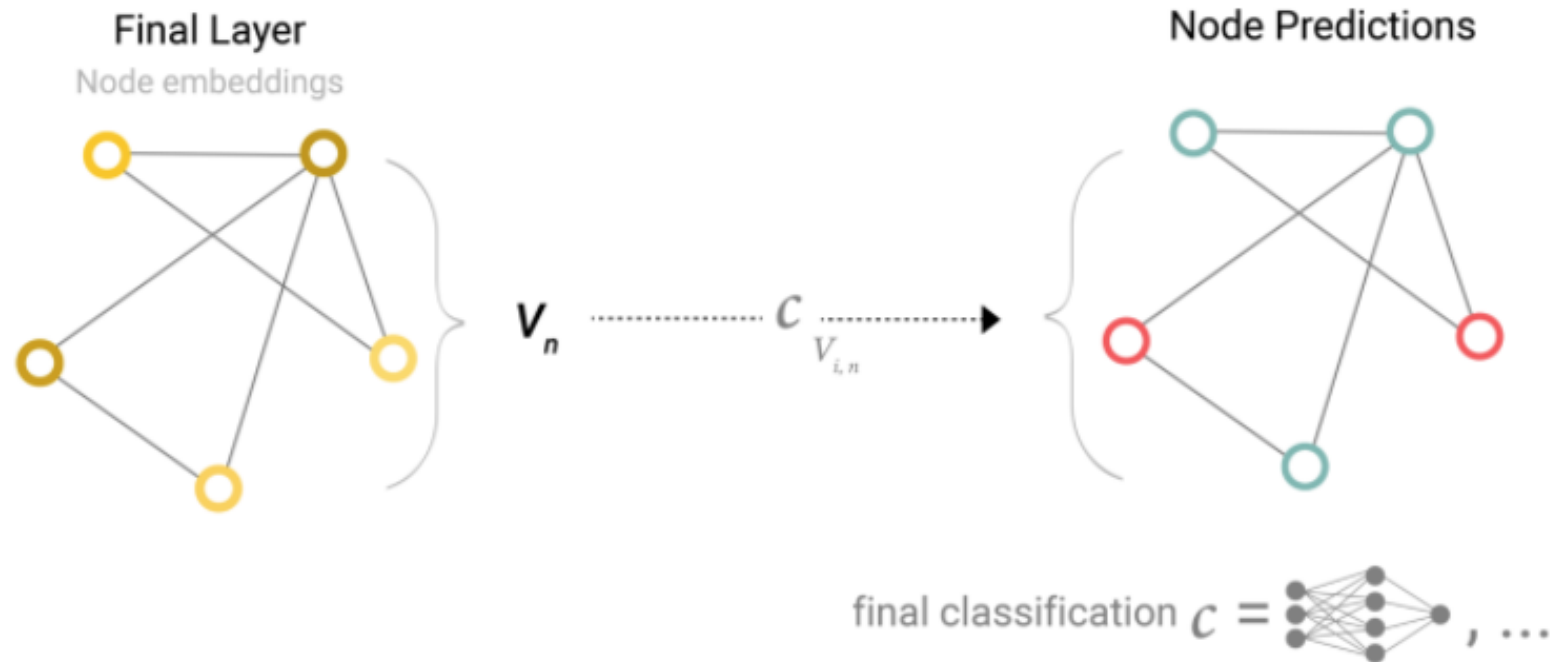


update function $f =$  , ...

Node Prediction

We built a simple GNN, but how do we make predictions in any of the tasks we described above?

If the task is to **make binary predictions on nodes**, and the graph already contains node information, **the approach is straightforward** — **for each node embedding, apply a linear classifier.**



GNN Predictions by Pooling Information

However, **it is not always so simple**. For instance, **you might have information in the graph stored in edges**, but no information in nodes, **but still need to make predictions on nodes**.

We need a way to collect information from edges and give them to nodes for prediction. **We can do this by *pooling***.

Pooling proceeds in two steps:

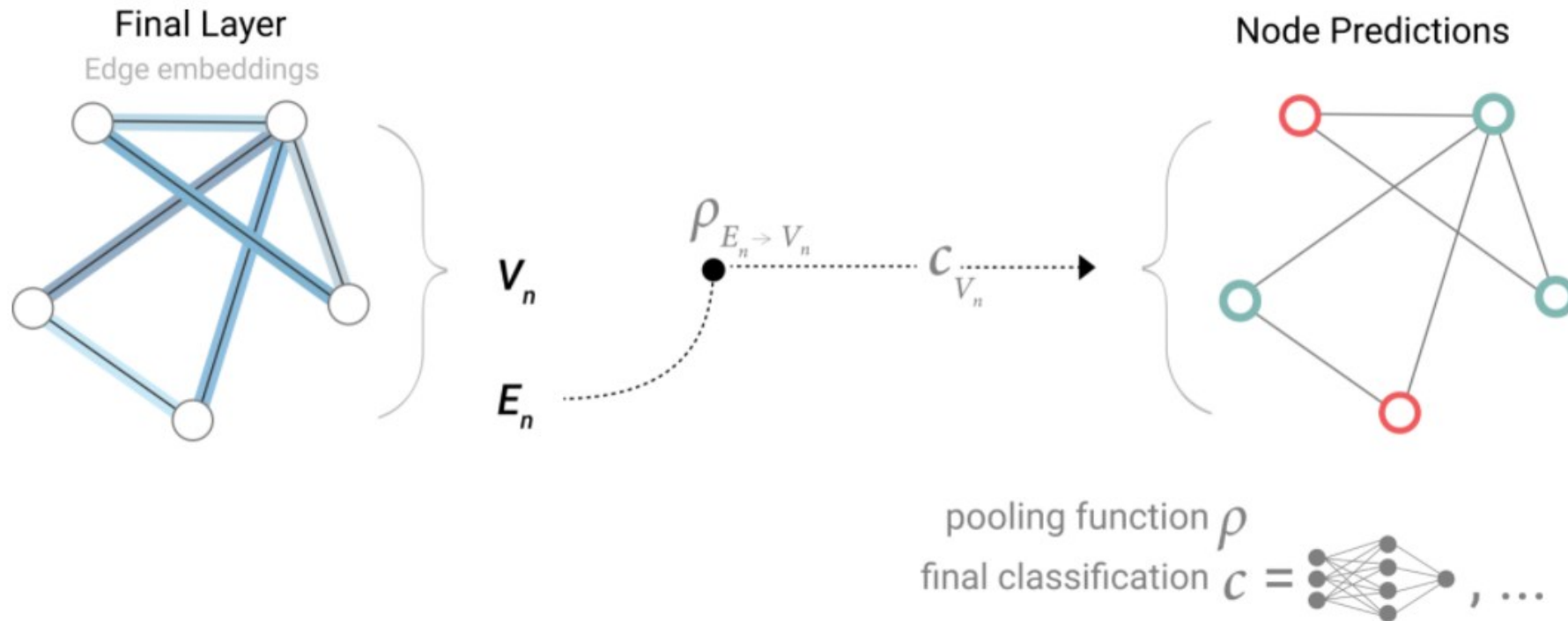
1. For each item to be pooled, ***gather*** each of their embeddings and concatenate them into a matrix.
2. The gathered embeddings are then ***aggregated***, usually via a sum or mean operation.

We represent the pooling operation by the letter ρ .

For example, we will use $\rho_{E_n \rightarrow V_n}$ to denote that **we are gathering information from edges to nodes**.

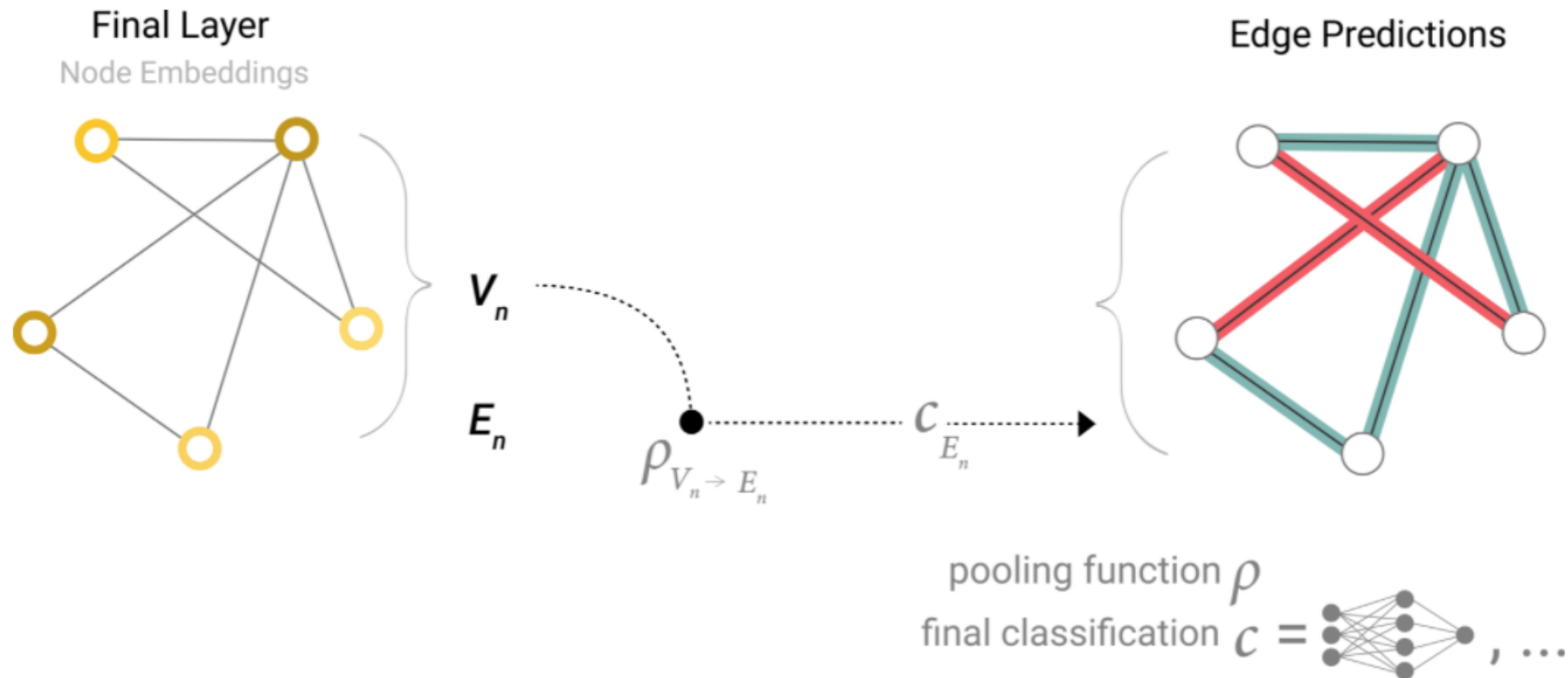
Node Predictions

So, if we **only have edge-level features**, and are trying to predict binary node information, we **can use pooling to route (or pass) information** to where it needs to go. **The model looks like this:**



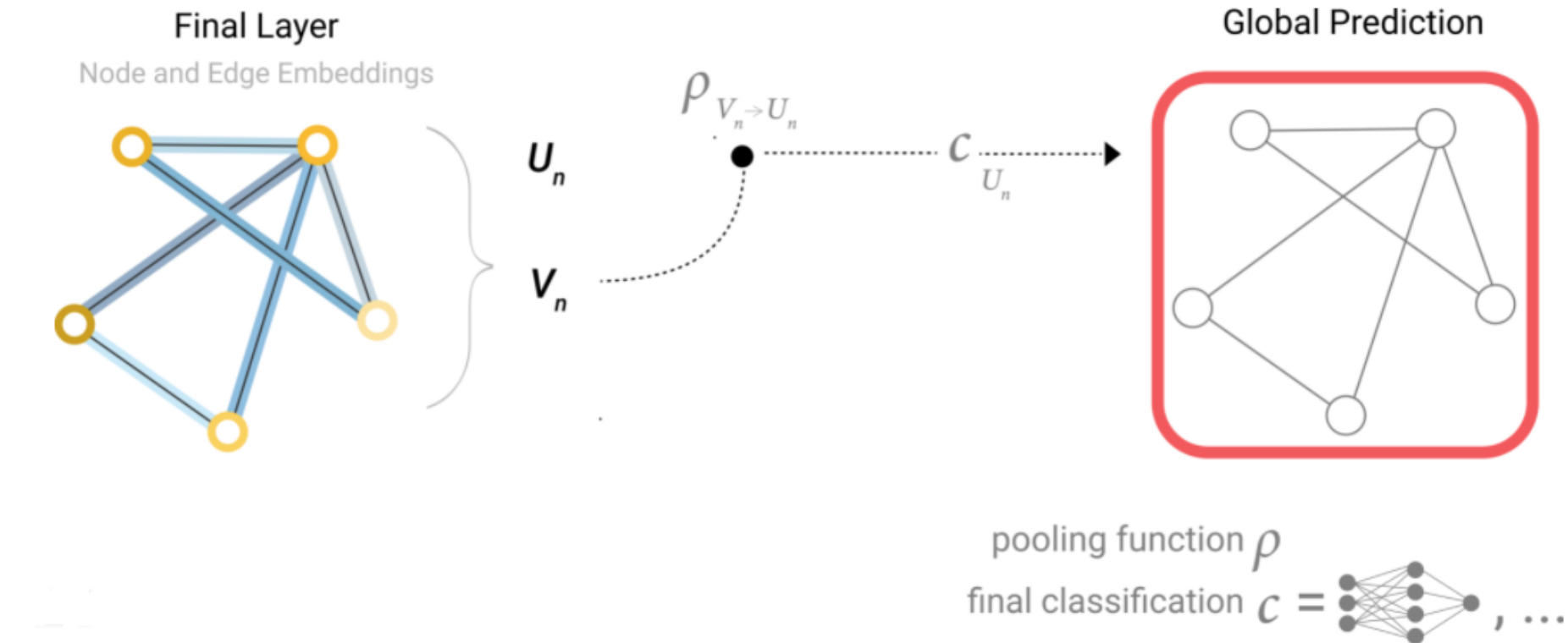
Edge Prediction

If we only have node-level features, and are trying to predict binary edge-level information, the model looks like this:



Global Property Predictions

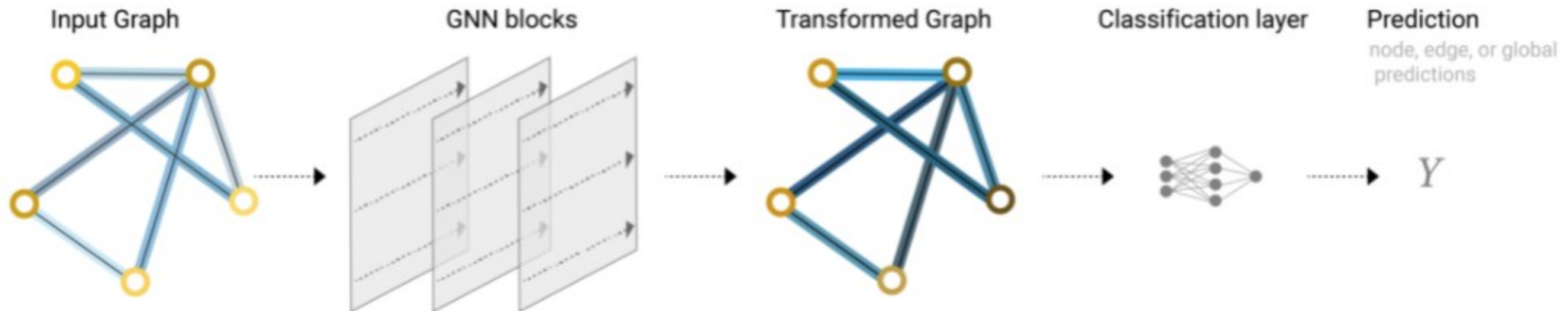
If we only have node-level features, and need to predict a binary global property, we need to gather all available node information together and aggregate them. This is similar to *Global Average Pooling* layers in CNNs. The same can be done for edges.



Overall Architecture

In our examples, the classification model/layer can easily be replaced with any differentiable models or adapted to multi-class classification using a generalized linear model.

Note that in this simplest GNN formulation, we're not using the connectivity of the graph at all inside the GNN layers. Each node is processed independently, as is each edge, as well as the global context. **We only use connectivity when pooling information for prediction.**



An end-to-end prediction task with a GNN model.

Passing Messages

Passing messages between parts of the graph

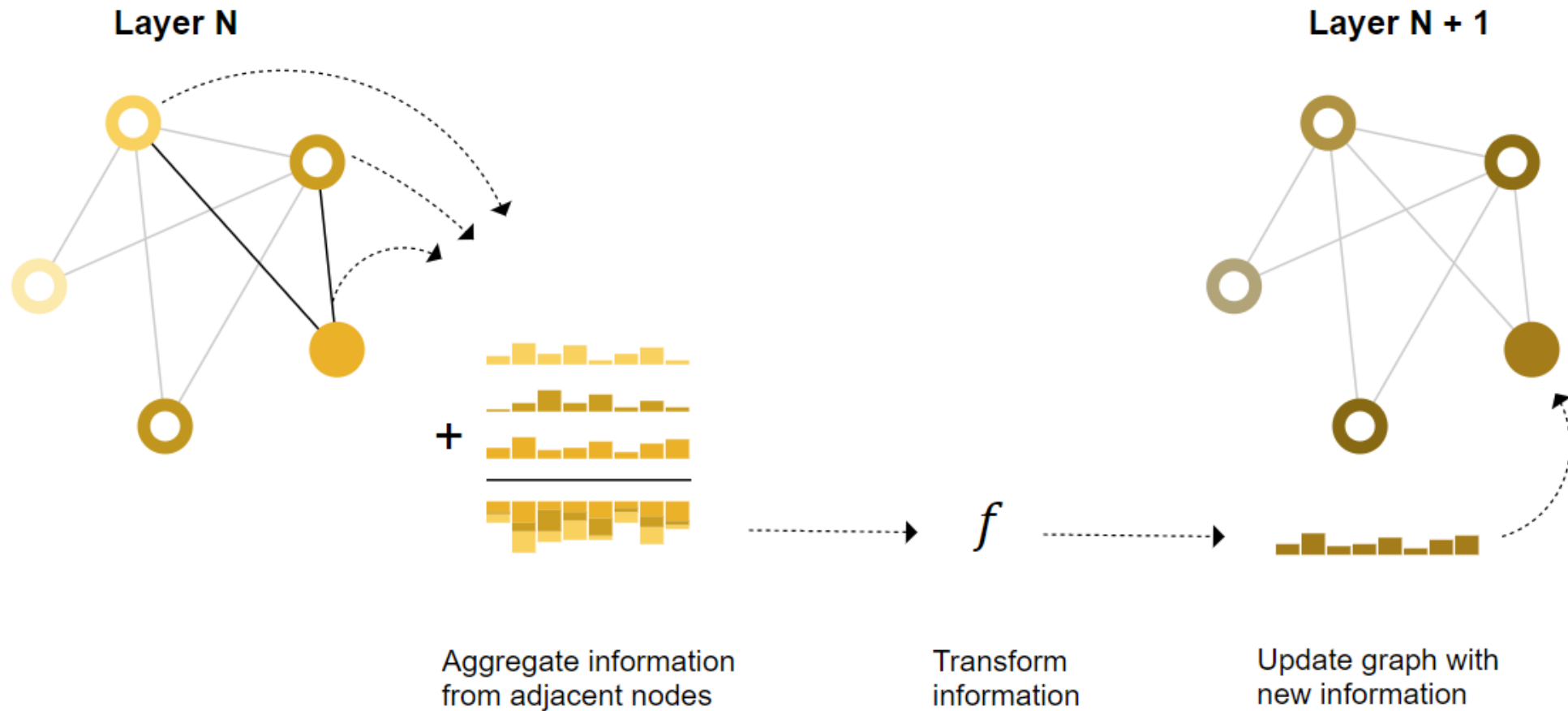
We could make more sophisticated predictions by using pooling within the GNN layer, in order to make our learned embeddings aware of graph connectivity. **We can do this using message passing, where neighboring nodes or edges exchange information and influence each other's updated embeddings.**

Message passing works in three steps:

1. For each node in the graph, **gather all the neighboring** node embeddings (or messages).
2. Aggregate **all messages via an aggregate function** (like sum).
3. All **pooled messages are passed through an update function**, usually a learned neural network.

Just as pooling can be applied to either nodes or edges, message passing can occur between either nodes or edges.

Passing messages between parts of the graph



Convolution

- Input I is a map of features (ex.: pixels of an image)
- There is a kernel K with trainable weights
- The kernel moves over the input map I: features propagate through near cells and are summed together
- Output map M is a summary of the input

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

0	1	2
2	2	0
0	1	2

12	12	17
10	17	19
9	6	14

12	12	17
10	17	19
9	6	14

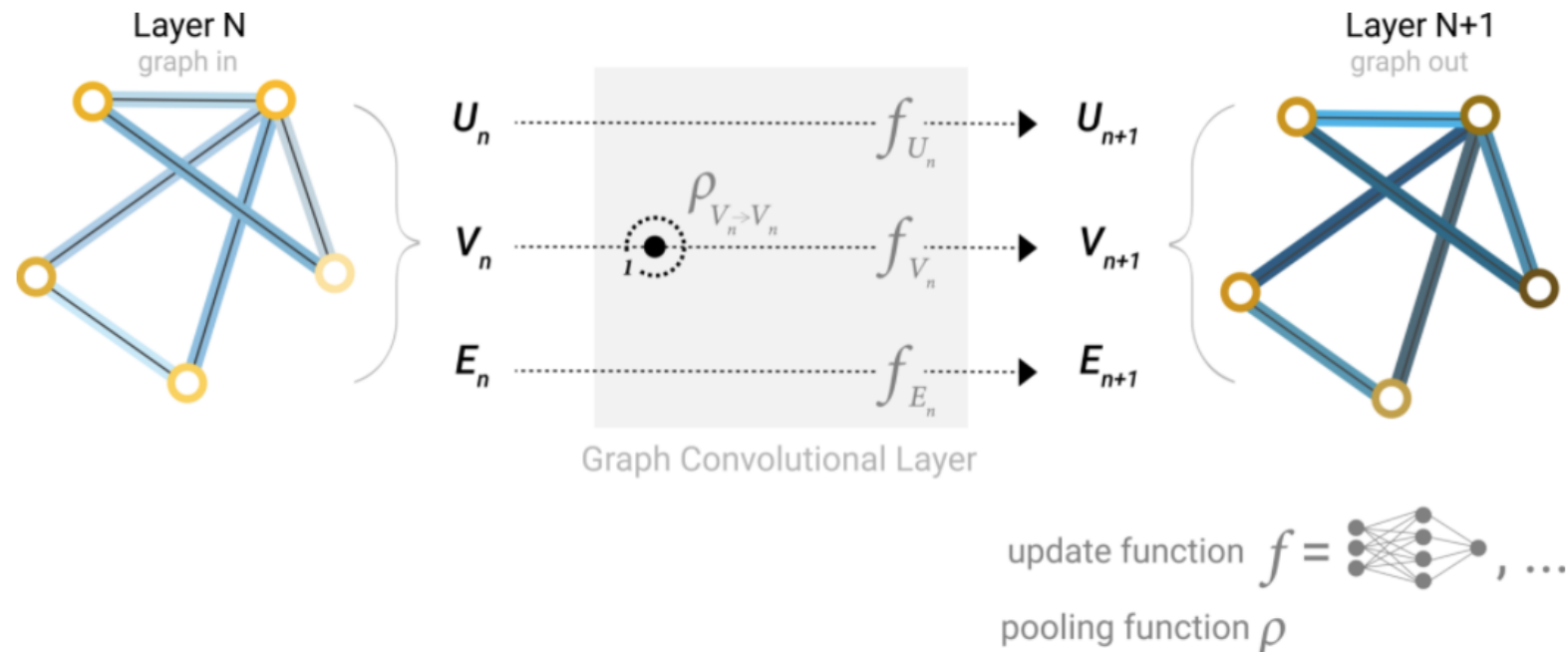
12	12	17
10	17	19
9	6	14

Passing messages and Convolution

In essence, **Message Passing and Convolution are operations to aggregate and process the information of an element's neighbors in order to update the element's value.** In graphs, the element is a **node**, and in images, the element is a **pixel**.

However, **the number of neighboring nodes in a graph can be variable**, unlike in an image where each pixel has a set number of neighboring elements. By stacking message passing GNN layers together, a node can eventually incorporate information from across the entire graph: **after three layers, a node has information about the nodes three steps away from it.**

Here the architecture diagram



Learning edge representations

Our dataset does not always contain all types of information (node, edge, and global context).

When we want to make a prediction on nodes, but our dataset only has edge information, **we showed above how to use pooling to route information from edges to nodes, but only at the final prediction step of the model.**

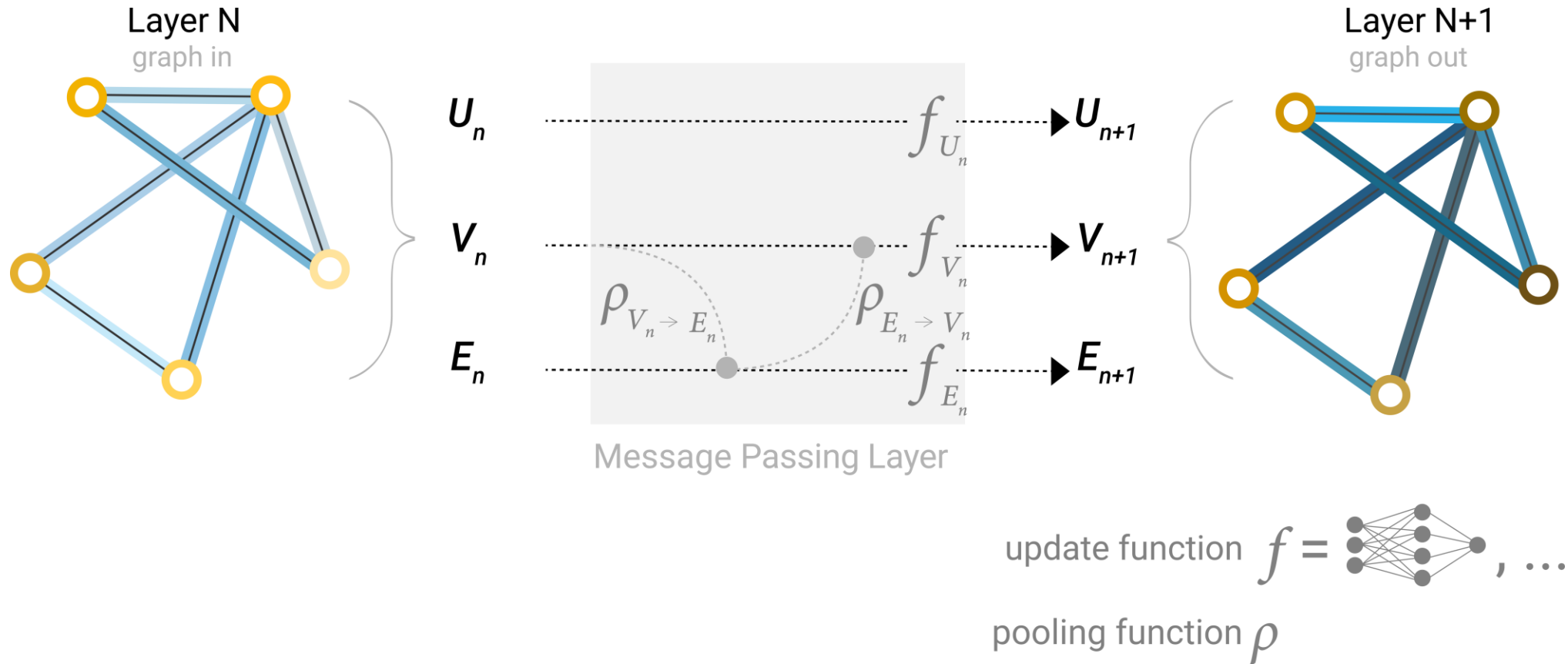
We can share information between nodes and edges within the GNN layer using message passing.

We can incorporate the information from neighboring edges in the same way we used neighboring node information earlier, by first pooling the edge information, transforming it with an update function, and storing it.

However, **the node and edge information stored in a graph are not necessarily the same size or shape. One way is to learn a linear mapping model** from the space of them edges to the space of nodes, and vice versa.

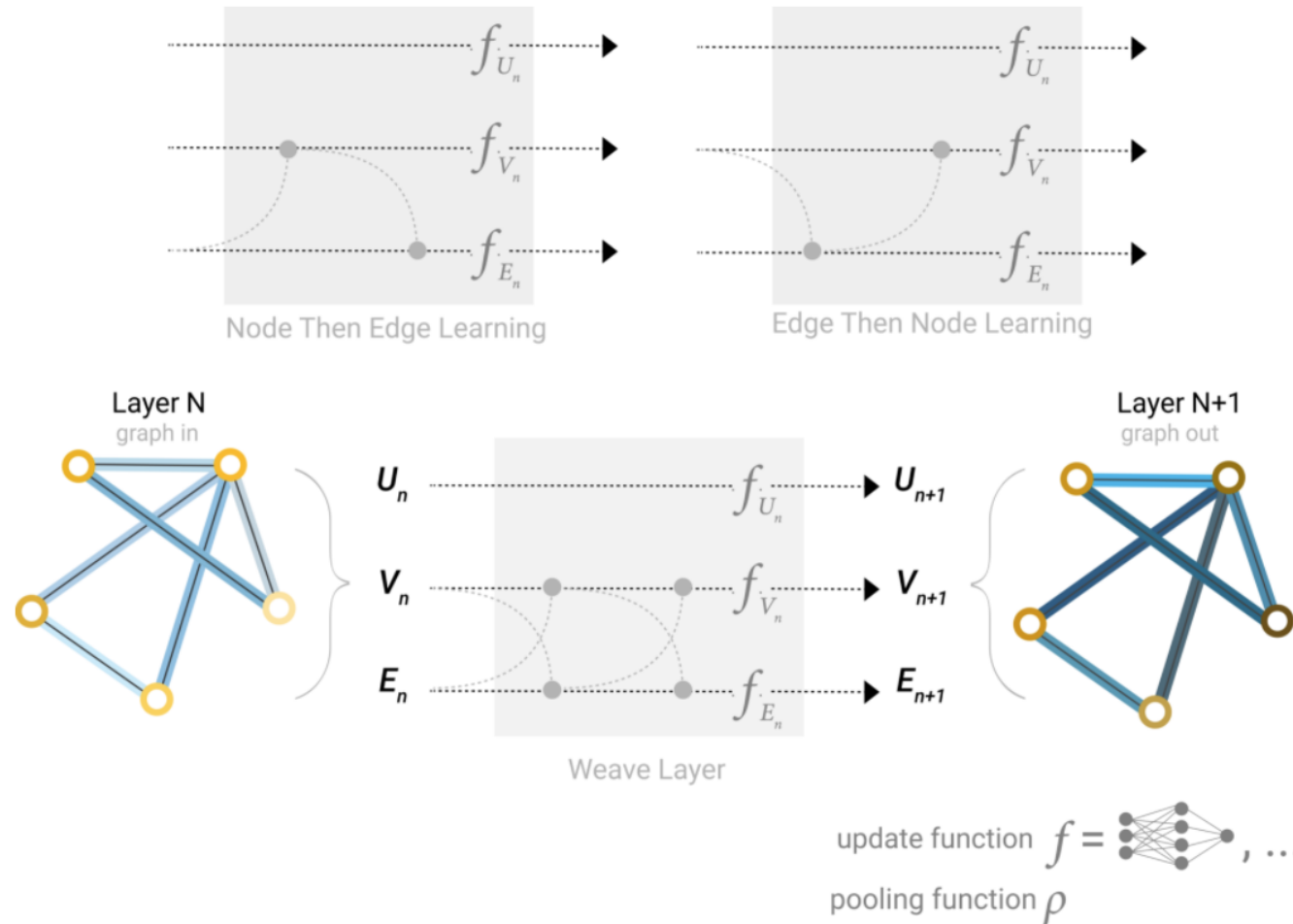


Learning edge representations



Architecture schematic for Message Passing layer

How to update node embedding and edge is an open are of research....



An Example

Graphs: definitions

- A graph is a set $G = (V, E)$ where V is the set of the N nodes $v_i \in V$ and $e_{i,j} \in E$ are the connecting edges
- Adjacency matrix $A \in \mathbb{R}^{N \times N}$ takes care of the connections
- Degree matrix $D_{ii} = \sum_j A_{ij}$ with the number of connections
- Feature matrix $X \in \mathbb{R}^{N \times C}$ where C is the dimension of the features vector

0	1	1	0	0
1	0	1	1	0
1	1	0	1	0
0	1	1	0	1
0	0	0	1	0

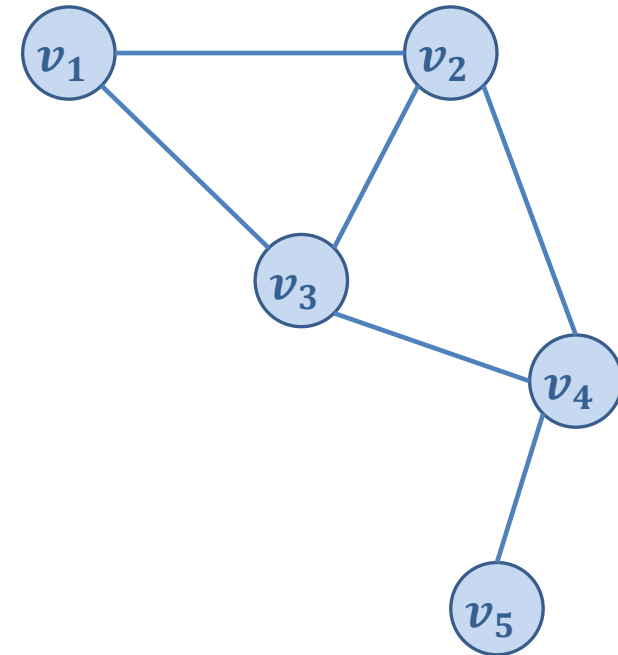
Adjacency A ($N \times N$)

2	0	0	0	0
0	3	0	0	0
0	0	3	0	0
0	0	0	3	0
0	0	0	0	1

Degree D ($N \times N$)

-1.1	2.5	0.6
3.1	-2.6	2.7
-0.3	1.7	-2.0
2.9	0.2	-1.9
2.8	-1.2	-0.9

Feature X ($N \times C$)



GCN: Graph Convolutional Networks (Kipf and Welling, 2016)

- Introduces Convolutional Neural Networks on graphs
- Consider a simple graph
 - $N=5$ nodes
 - $C=3$ components features vectors

0	1	1	0	0
1	0	1	1	0
1	1	0	1	0
0	1	1	0	1
0	0	0	1	0

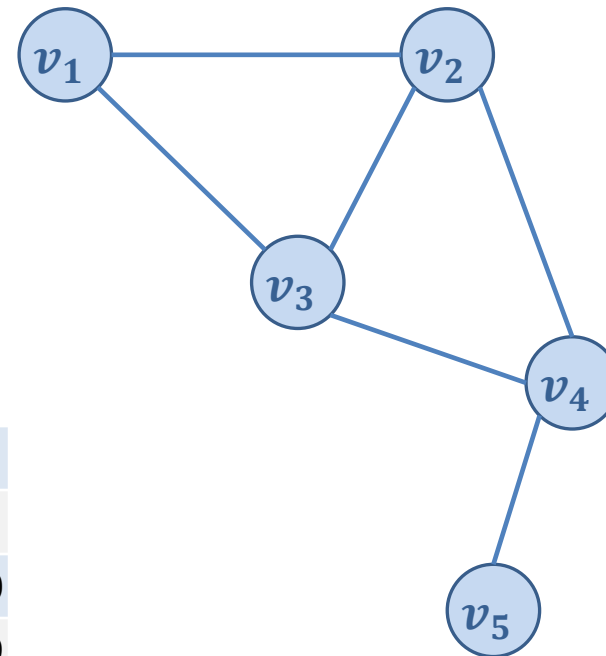
Adjacency A ($N \times N$)

2	0	0	0	0
0	3	0	0	0
0	0	3	0	0
0	0	0	3	0
0	0	0	0	1

Degree D ($N \times N$)

-1.1	2.5	0.6
3.1	-2.6	2.7
-0.3	1.7	-2.0
2.9	0.2	-1.9
2.8	-1.2	-0.9

Feature X ($N \times C$)



GCN: propagation of features

- First attempt: $X' = A \cdot X$

0	1	1	0	0
1	0	1	1	0
1	1	0	1	0
0	1	1	0	1
0	0	0	1	0

Adjacency A

\times

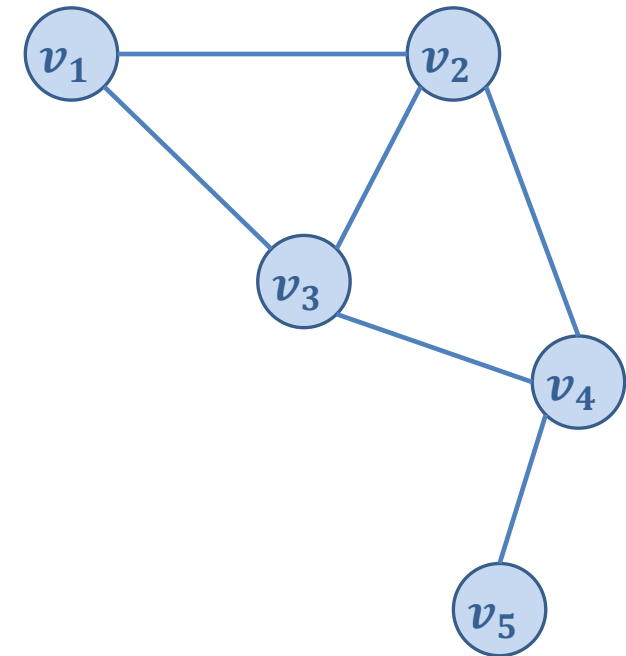
-1.1	2.5	0.6
3.1	-2.6	2.7
-0.3	1.7	-2.0
2.9	0.2	-1.9
2.8	-1.2	-0.9

Feature X

$=$

2.8	-0.9	0.7
1.5	4.4	-3.3
4.9	0.1	1.4
5.6	-2.1	-0.2
2.9	0.2	-1.9

Feature X'



- Two issues:
 - The node i doesn't receive its own features
 - Summing features could lead to explosion for high degrees values

GCN: propagation of features, self adjacency

- Fix for the first issue: add self adjacency, $\tilde{A} = A + \lambda I$, $X' = \tilde{A} \cdot X$

0	1	1	0	0
1	0	1	1	0
1	1	0	1	0
0	1	1	0	1
0	0	0	1	0

 $+$

λ	0	0	0	0
0	λ	0	0	0
0	0	λ	0	0
0	0	0	λ	0
0	0	0	0	λ

 $=$

λ	1	1	0	0
1	λ	1	1	0
1	1	λ	1	0
0	1	1	λ	1
0	0	0	1	λ

Adjacency A

λI

Adjacency \tilde{A}

1	1	1	0	0
1	1	1	1	0
1	1	1	1	0
0	1	1	1	1
0	0	0	1	1

 \times

-1.1	2.5	0.6
3.1	-2.6	2.7
-0.3	1.7	-2.0
2.9	0.2	-1.9
2.8	-1.2	-0.9

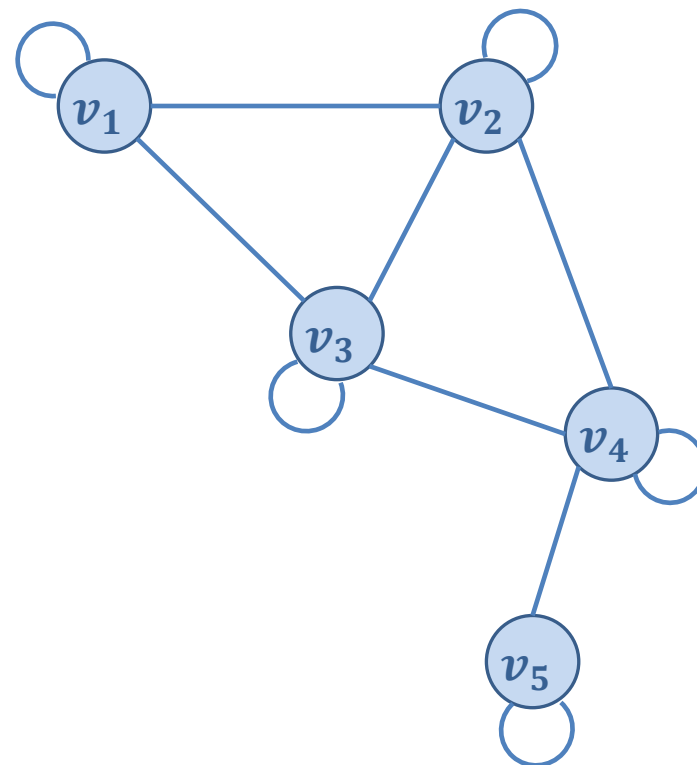
 $=$

1.7	1.6	1.3
4.6	1.8	-0.6
4.6	1.8	-0.6
8.5	-1.9	-2.1
5.7	-1	-2.8

$\tilde{A} (\lambda = 1)$

X

X'



GCN: propagation of features, averaging signals

- Fix for the second issue: divide by the degrees of the node (mean over neighbour nodes), $X' = (\tilde{D}^{-1} \cdot \tilde{A}) \cdot X$

1	1	1	0	0
1	1	1	1	0
1	1	1	1	0
0	1	1	1	1
0	0	0	1	1

 \tilde{A}
→

3	0	0	0	0
0	4	0	0	0
0	0	4	0	0
0	0	0	4	0
0	0	0	0	2

 \tilde{D}
→

1/3	0	0	0	0
0	1/4	0	0	0
0	0	1/4	0	0
0	0	0	1/4	0
0	0	0	0	1/2

 \tilde{D}^{-1}

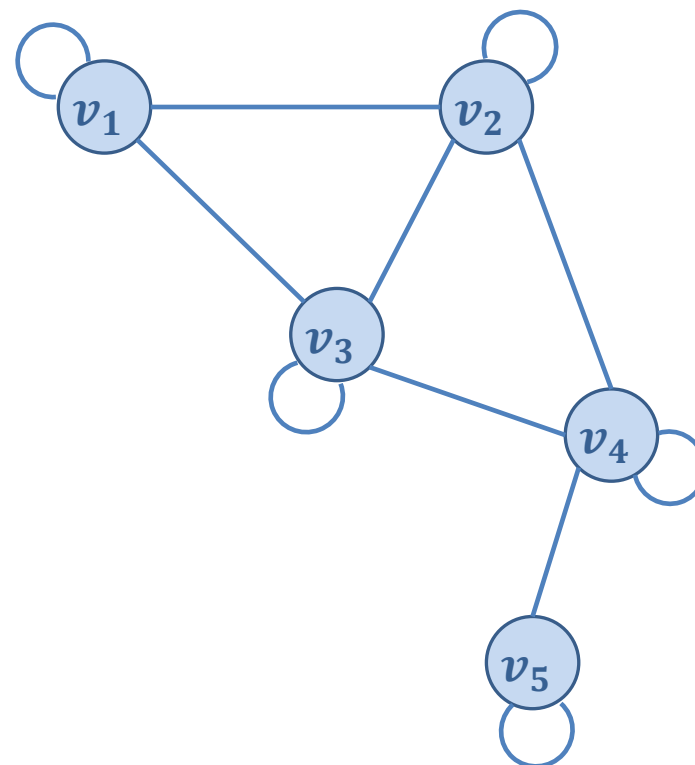
1/3	0	0	0	0
0	1/4	0	0	0
0	0	1/4	0	0
0	0	0	1/4	0
0	0	0	0	1/2

 \tilde{D}^{-1}
×

1	1	1	0	0
1	1	1	1	0
1	1	1	1	0
0	1	1	1	1
0	0	0	1	1

 \tilde{A}
=

1/3	1/3	1/3	0	0
1/4	1/4	1/4	1/4	0
1/4	1/4	1/4	1/4	0
0	1/4	1/4	1/4	1/4
0	0	0	1/2	1/2

 $\tilde{D}^{-1} \cdot \tilde{A}$


GCN: propagation of features, weighted average

- Further step: we want averages weighted with the degrees of the neighbouring

nodes: $X' = (\tilde{D}^{-1} \cdot \tilde{A}) \cdot \tilde{D}^{-1} \cdot X$

1/3	1/3	1/3	0	0
1/4	1/4	1/4	1/4	0
1/4	1/4	1/4	1/4	0
0	1/4	1/4	1/4	1
0	0	0	1/2	1/2

 \times

1/3	0	0	0	0
0	1/4	0	0	0
0	0	1/4	0	0
0	0	0	1/4	0
0	0	0	0	1/2

 $=$

1/9	1/12	1/12	0	0
1/12	1/16	1/16	1/16	0
1/12	1/16	1/16	1/16	0
0	1/16	1/16	1/16	1/16
0	0	0	1/8	1/4

$\tilde{D}^{-1} \cdot \tilde{A}$ \tilde{D}^{-1} $(\tilde{D}^{-1} \cdot \tilde{A}) \cdot \tilde{D}^{-1}$

1/9	1/12	1/12	0	0
1/12	1/16	1/16	1/16	0
1/12	1/16	1/16	1/16	0
0	1/16	1/16	1/16	1/16
0	0	0	1/8	1/4

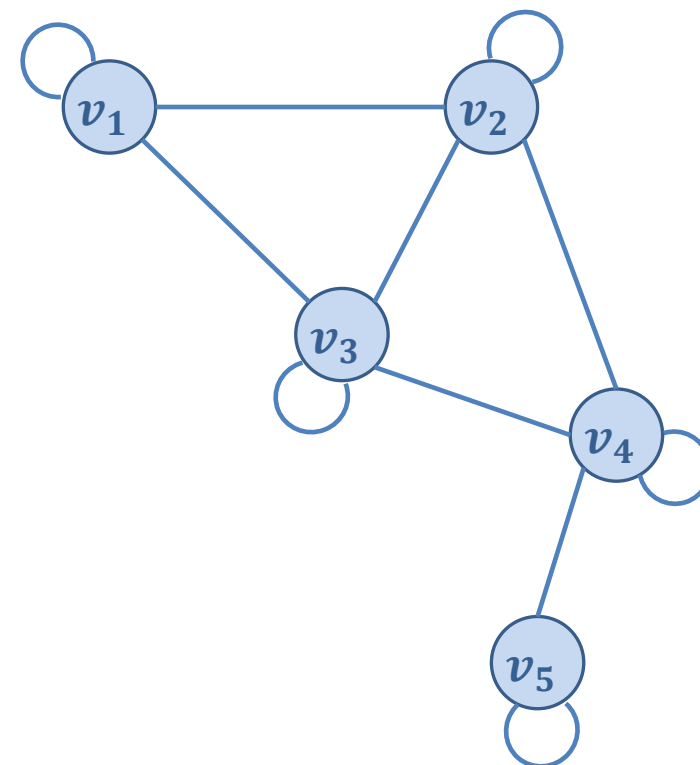
 \times

-1.1	2.5	0.6
3.1	-2.6	2.7
-0.3	1.7	-2.0
2.9	0.2	-1.9
2.8	-1.2	-0.9

 $=$

0.1	0.2	0.1
0.3	0.2	0.0
0.3	0.2	0.0
0.7	-0.2	-0.2
1.1	-0.3	-0.5

$(\tilde{D}^{-1} \cdot \tilde{A}) \cdot \tilde{D}^{-1}$ X X'



GCN: propagation of features, normalization

- We performed average two times, so normalize:

$$X' = (\tilde{D}^{-1/2} \cdot \tilde{A}) \cdot \tilde{D}^{-1/2} \cdot X$$

1/3	0	0	0	0
0	1/4	0	0	0
0	0	1/4	0	0
0	0	0	1/4	0
0	0	0	0	1/2

\tilde{D}^{-1}



1/√3	0	0	0	0
0	1/2	0	0	0
0	0	1/2	0	0
0	0	0	1/2	0
0	0	0	0	1/√2

$\tilde{D}^{-1/2}$



1/9	√3/2	√3/2	0	0
√3/2	1/4	1/4	1/4	0
√3/2	1/4	1/4	1/4	0
0	1/4	1/4	1/4	√2/2
0	0	0	√2/2	1/2

$(\tilde{D}^{-1/2} \cdot \tilde{A}) \cdot \tilde{D}^{-1/2}$

1/9	√3/2	√3/2	0	0
√3/2	1/4	1/4	1/4	0
√3/2	1/4	1/4	1/4	0
0	1/4	1/4	1/4	√2/2
0	0	0	√2/2	1/2

$(\tilde{D}^{-1/2} \cdot \tilde{A}) \cdot \tilde{D}^{-1/2}$



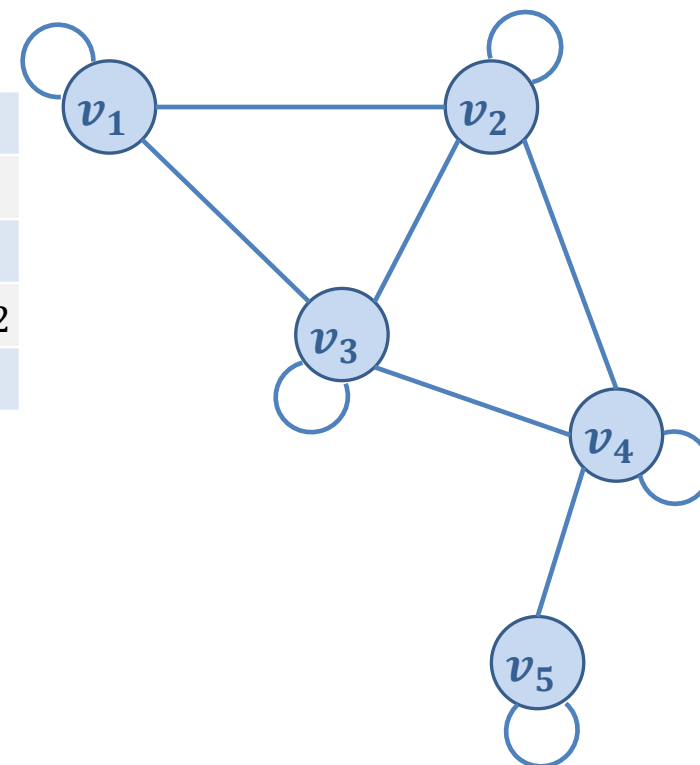
-1.1	2.5	0.6
3.1	-2.6	2.7
-0.3	1.7	-2.0
2.9	0.2	-1.9
2.8	-1.2	-0.9

X



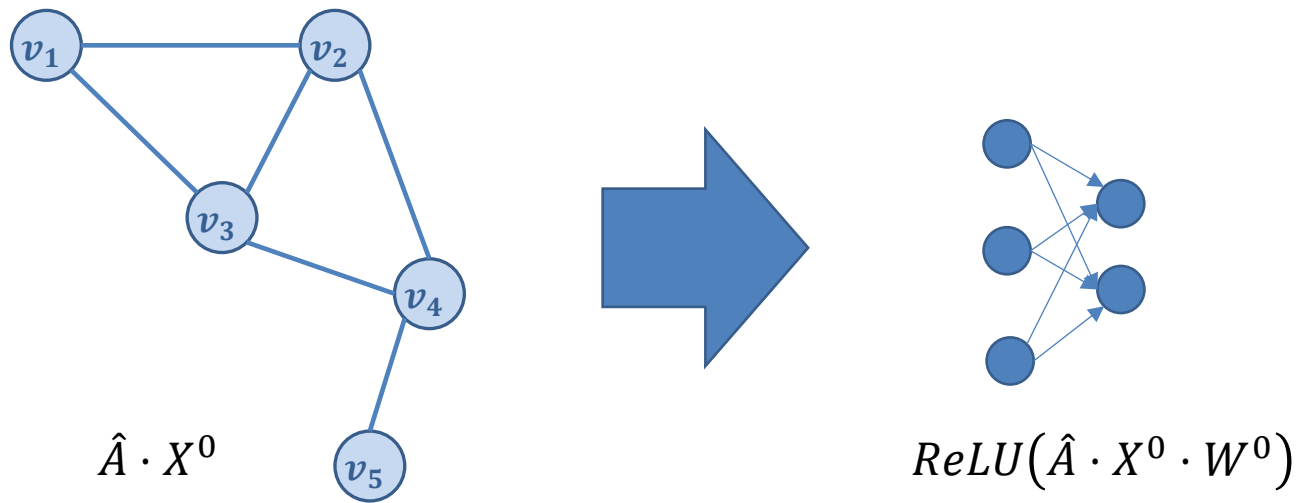
0.4	0.6	0.4
1.1	0.5	-0.1
1.1	0.5	-0.1
2.4	-0.6	-0.6
2.4	-0.5	-1.1

X'



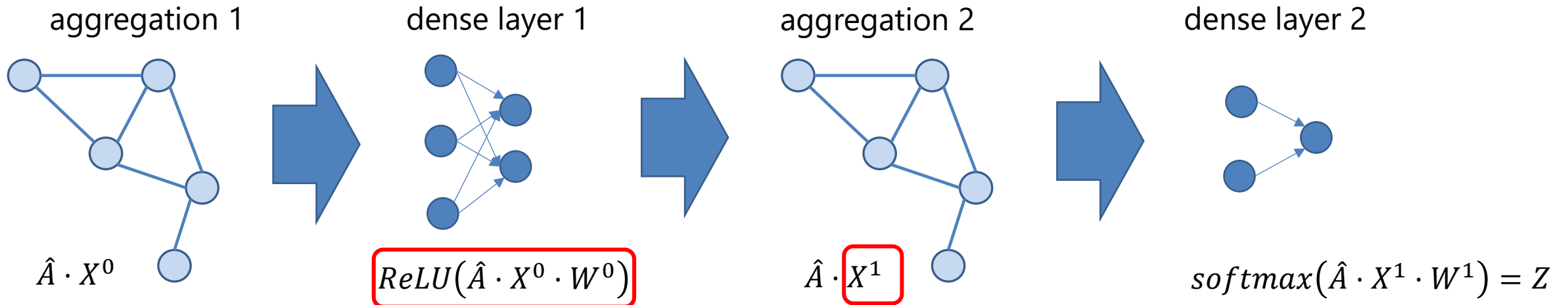
GCN: pass through Dense Layer

- Put: $\hat{A} = (\tilde{D}^{-1/2} \cdot \tilde{A}) \cdot \tilde{D}^{-1/2}$
- Finally, aggregated features are passed through a Dense Layer: this is the *GCN model*.



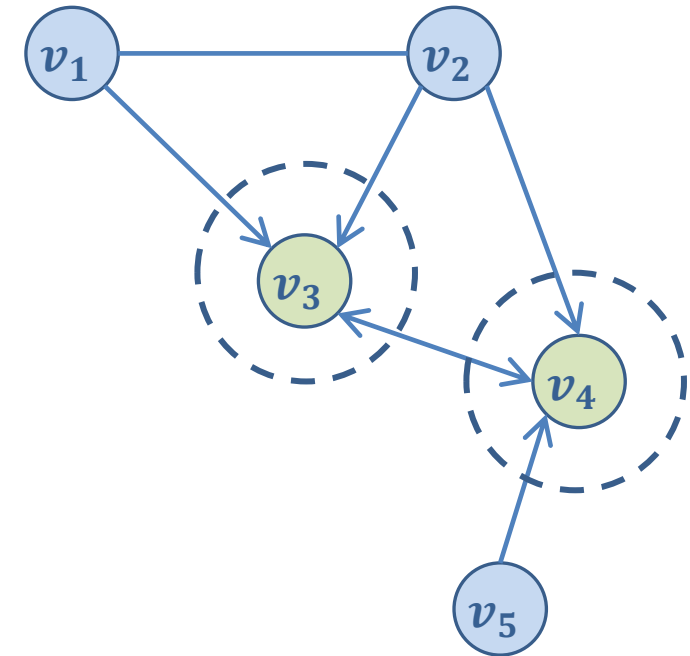
GCN: pass through Dense Layers

- Many GCN can be stacked together (here 2)
- Output of the first layer is input of the second
- Dimensions: $\hat{A}_{N \times N}$, $X^0_{N \times C}$, $X^1_{N \times H}$, $W^0_{C \times H}$, $W^1_{H \times L}$, $Z_{N \times L}$



GCN: network depth

- **2 GCN layers means that each node receives signals from nodes at distance 2**
- Look at nodes 3 and 4: **first GCN processes signals from neighbours**
- Second GCN propagates updated node features through neighbours and aggregates again, but neighbours have already aggregated their neighbours in turn
- *Number of layers means network depth*



Reference

- <https://distill.pub/2021/gnn-intro/>